유비쿼터스 게임 플랫폼에서 멀티스레딩을 위한 가상기계의 설계 및 구현

최홍석*, 이양선*
*서경대학교 컴퓨터공학과
e-mail:{lagnaroc, yslee}@skuniv.ac.kr

Design and Implementation of a Virtual Machine for MultiThread in the Ubiquitous Game Platform

Hong-Suck Choi*, Yang-Sun Lee*
*Dept of Computer Engineering, SeoKyeong University

요 약

본 연구팀은 유비쿼터스 환경에서 다양한 분야의 콘텐츠를 보다 쉽게 개발하고 실행할 수 있는 통합 소프트웨어 개발 솔루션인 유비쿼터스 게임 플랫폼(Ubiquitous Game Platform)을 개발하였다. 유비쿼터스 게임 플랫폼은 가상기계 방식으로써 플랫폼에 독립적어서 유비쿼터스 환경의 기기에 탑재가 수월한 장점을 가진다. 본 논문에서는 유비쿼터스 게임 플랫폼에서 다양한 콘텐츠의 실행과 멀티 스레딩을 지원하는 유비쿼터스 가상기계(u-VM)를 설계하고 구현하였다. MS의 .NET 플랫폼과 SUN의 JVM이 C/C++나 Java 언어 하나만을 지원하는데 반해 본 연구에서 제시한 u-VM은 다양한 유비쿼터스 기기에 탑재되어 C/C++, Java 언어로 작성된 다양한 모든 종류의 어플리케이션을 실행한다. u-VM은 SEF(Standard Executable Format) 실행 파일을 입력으로 받아 실행하며, SEF 로더와 인터프리터, 내장 라이브러리, 실행 환경으로 구성되어 있다. 실행 환경에서는 메모리를 관리하고 예외를 처리하며 스레드 스케줄러를 통해 멀티스레딩 기능을 제공 한다.

1. 서론

현대 사회의 사람들에게 유비쿼터스는 매우 친숙한 단어로 자리 잡았다. 사람들은 유비쿼터스 환경의 임베디드 기기들을 한 개 이상씩은 지니고 다니고 있으며 임베디드 기기들은 어느새 우리 삶의 중요한 요소가 되었다.

그만큼 대중들에게 유비쿼터스 환경의 임베디드 기술은 널리 알려진 기술이 되었으며 최근 가장 큰 이슈가 되고 있는 연구 과제가 되었다.

그러나 이런 임베디드 기기들의 H/W적인 요소들은 강력하게 발전하고 있는 반면 각 기기들을 제어하거나 H/W 자원을 효율적으로 이용할 수 있게 하는 S/W 콘텐츠와 그의 개발 환경은 상대적으로 빈약한 실정이다.

현재 임베디드 기기의 콘텐츠 제작 환경은 같은 콘텐츠라도 목적 기기에 따른 수정이 필요하거나 심지어는 재개발 까지 필요한 실정이다. 또한 PC환경의 개발 언어들의 서브셋만을 지원하는 등의 경우까지 존재하며 개발 제약 또한 심하다.

이런 비효율 적인 환경으로 인해 임베디드 기기의 콘텐츠의 개발은 많은 어려움을 겪고 있다.

본 논문에서 제안하는 멀티 스레드 환경을 지원하는 가상기계인 u-VM(ubiquitous-Virtual Machine)은 본 연구팀이 개발한 유비쿼터스 게임 플랫폼의 실행을 담당하며 유비쿼터스 게임 플랫폼은 임베디드 기기의 어려운 콘텐츠 제작 환경을 개선해 주는 방법을 제시한다.

u-VM은 가상기계기반의 S/W로서 플랫폼에 독립적이어서 프로세서 및 운영체제가 다양하고 변경이 잦은 임베디드 기기 환경에 적합하며 C, C++, Java 언어들을 모두 수용할 수 있는 중간코드를 입력 받음으로써 언어에 구애받지 않는 개발 환경을 제공한다.[5]

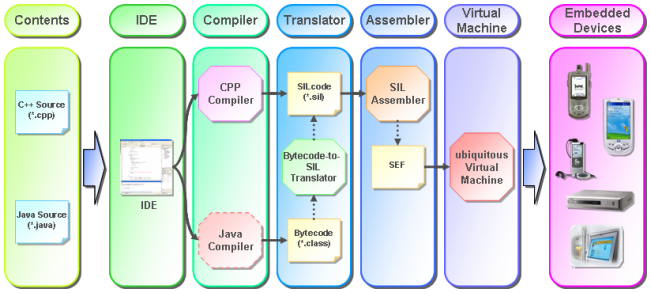
본 연구는 문화관광부 및 한국문화콘텐츠진흥원의 '06문화콘텐츠기술(CT)개발지원사업의 연구결과로 수행되었음

2. 관련 연구

2.1 유비쿼터스 게임 플랫폼

유비쿼터스 게임 플랫폼은 본 연구팀이 개발한 유비쿼터스 환경의 임베디드 기기들을 위한 게임 플랫폼 이다.

중간 언어를 사용하는 가상 기계 기반의 플랫폼이어서 여러 임베디드 기기에 독립적으로 수행이 가능하며 중간 언어인 SAF는 순차적 언어인 C와 객체지향 언어인 C++, Java를 모두 수용할 수 있어서 세 가지 언어를 모두 사용할 수 있는 장점을 가진다.



(그림 1) 유비쿼터스 게임 플랫폼의 구조

유비쿼터스 게임 플랫폼의 구조도는 (그림 1)과 같다. 유비쿼터스 게임 플랫폼은 컴파일/번역 부인 전단부와 실행 부인 후단부로 나뉜다.

전단 부는 u-VM이 실행 가능한 SEF(Standard Executable Format)를 출력하는 부분으로써 IDE를 이용하여 C , C++, Java 언어를 이용해 콘텐츠를 제작할 수 있는 환경을 제공해 준다.

제작된 콘텐츠들은 C, C++의 경우 CPP 컴파일러를 통해 순차지향언어와 객체지향 언어를 모두 수용하는 중간언어 형식인 SAF를 출력으로 내고, Java의 경우 Sun사의 Java 컴파일러를 통해 바이트 코드를 낸 후 SAF로 출력하기 위해 Bytecode-to-SAF 번역기를 이용하여 SAF를 출력한다.

출력된 SAF는 로더와 링커의 역할을 하는 SAF 어셈블러를 통해 u-VM이 실행 가능한 SEF으로 변환된다.

이렇게 출력된 SEF는 후단부에 있는 u-VM에서 실행을 하게 된다. VM이 탑재된 IDE의 경우 에뮬레이터에서의 실행을 담당하게 되며 기기 내 탑재된 u-VM의 경우 실제 임베디드 기기 내에서 실행이 된다.[7]

2.2 SAF 어셈블러

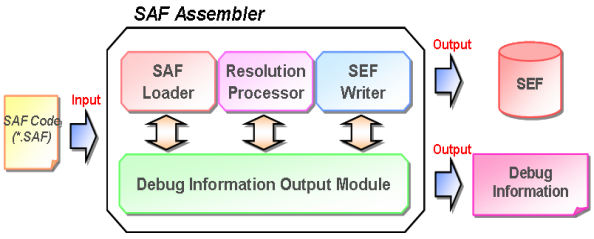
기존 u-VM은 어셈블러의 내장으로 SAF를 입력으로 받아 실행하였다. 그러나 텍스트 형식의 SAF 포맷을 실행파일 포맷으로 쓰기엔 소스코드의 보안에 취약한 문제를 가진다.

그리고 u-VM의 내장 어셈블러에선 링킹기능이 상대적으로 약하여 여러 개의 소스 코드들을 컴파일 시 각 소스들을 링킹 하기에 어려움이 따랐다.

이런 문제점을 해결하기 위해 본 연구팀은 따로 SAF 어셈블러를 제작하여 바이너리 형식인 SEF 포맷을 출력하여 보안상의 문제점을 해결하였으며 여러 개의 소스코드를 한 대 묶는 링킹 과정을 강화 하였다.

어셈블러의 구조도는 (그림 2)와 같다. SAF 어셈블러는 SAF Loader를 이용하여 SAF의 정보를 메모리에 적재한다. 그 정보를 실제 실행에 쓰이는 정보와 디버깅에 쓰이는 정보를 구분하여 디버깅 정보는 Debug Information Output Module에서 처리 하여 디버깅 정보를 뿌리고, 실행에 쓰이는 정보는 Resolution Processor를 통해 각 심벌들을 바인딩 하고 SEF 포맷에 맞추어 바이너리 형식으로 뿌리게 된다.

이렇게 출력된 SEF파일은 u-VM의 입력으로 쓰이게 되며 디버거 정보는 나중에 추가될 모듈인 디버거의 정보로 쓰이게 될 것이다.



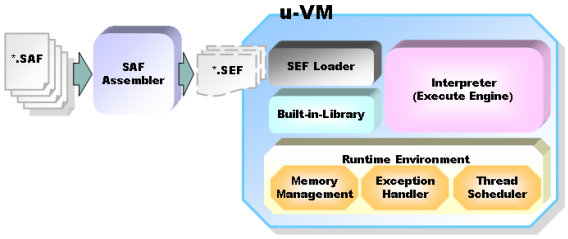
(그림 2) SAF 어셈블러의 구조도

3. u-VM(ubiquitous-Virtual Machine)

3.1 u-VM의 구조

u-VM은 순차지향 언어와 객체지향 언어를 모두 수용하는 중간언어 형식인 SAF에서 SAF어셈블러를 통해 나온 SEF를 입력으로 받아 실행되는 스택 기반의 머신이다.

u-VM의 구성은 (그림 3)과 같이 SEF 로더와 인터프리터(Execute Engine), 내장 라이브러리(Built-in-Library), 실행환경(Runtime Environment)으로 이루어져 있다.



(그림 3) u-VM의 구조

SAF 어셈블러를 통해 나온 SEF는 SEF Loader를 통해 의사코드 정보나 연산 코드 정보를 코드 영역(Code Area), 상수풀(Constant Pool)이란 메모리에 적재하는 역할을 한다.

u-VM은 이렇게 수집한 정보를 토대로 실행환경에서 메모리 관리, 예외처리, 스레드 스케줄링 역할을 한다.

메모리 관리자(Memory Management)는 프로그램 실행에 필요한 메모리 영역을 Heap, Local Area, Global Area로 나누어 관리를 하며 스레드 스케줄러(Thread Scheduler)는 여러 개의 스레드를 생성, 소멸, 제어하는 역할을 하여 Multi Tasking이 가능한 환경을 제공하여준다.

예외 처리기(Exception Handler)는 프로그램 상의 예외를 처리하여 주는 역할을 하여 준다.

내장 라이브러리는 프로그램 수행에 필요한 API를 가상 기계 내에서 구현한 부분으로써 현재 ANSI C 라이브러리와 Java 라이브러리, 그래픽 라이브러리의 일부를 지원하고 있으며 손쉽게 다른 API의 추가가 가능하도록 설계 되어 있다.

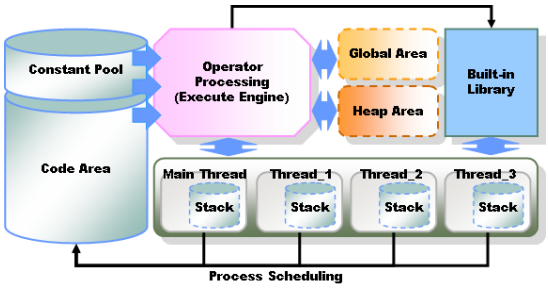
인터프리터는 u-VM의 핵심적인 기능을 하는 모듈로써 메모리에 적재되어 있는 정보들을 디스패치 하여 각 연산 코드들을 실제로 실행하는 역할을 하고 있다.

인터프리터에서 실제 실행하는 코드들은 실행 환경 내에서 제공하는 메모리를 참조로 하여 실행되며 대부분 스택기반의

연산을 통해 실행 한다.[7]

3.2 멀티 스레드 환경의 u-VM의 설계 및 구현

(그림 4)는 멀티 스레드 환경의 u-VM 구현 모델을 도식화 한 것이다. 연산자를 처리하는 실행 엔진에서는 상수 풀(Constant Pool)과 코드 영역(Code Area)의 정보를 통해 실행을 한다.



(그림 4) u-VM의 구현 모델

상수 풀 정보는 글로벌 영역에 적재되어 사용되며 코드영역에서의 연산 코드에 따라 힙 영역과 로컬 영역에의 적재가 이루어진다.

로컬 영역은 Activation Record와 Operation Stack, Display Vector로 구성된 스택 구조로 이루어져 메모리를 관리하며 스레드의 생성 시 마다 같이 생성 된다.[7]

적재된 정보들은 <Base> <Offset> 정보를 통해 접근이 가능하며 이 정보들은 PC(Program Counter)에 따라 연산 코드와 의사 코드 정보를 패치 하여 POP, PUSH 과정을 통하여 실행 한다.

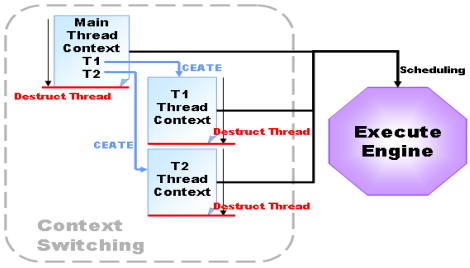
일반 프로그램의 실행은 단일 스레드의 수행으로 이루어지나 멀티스레딩을 사용하는 프로그램은 여러 개의 스레드의 수행으로 이루어진다.

각 스레드는 스택 구조인 로컬 영역을 가지게 된다. 그럼으로써 각 스레드는 각자의 스택에 의해 로컬 영역을
처리하며 글로벌 영역과 힙 영역은 서로 공유 한다.[2]

2개 이상의 스레드가 실행될 때에는 각 스레드간의 제어 흐름을 통제해야 하는데 u-VM은 스레드 스케줄러에 의해 처리 된다. u-VM은 라운드 로빈 방식의 스케줄링 방식[4]을 채택하여 구현되었다.

(그림 5)처럼 메인 스레드에서 T1 스레드를 생성하는 순간 스레드는 2개가 되어 일정 시간동안 서로 문맥 교환 하며 2개의 스레드가 실행되며 메인 스레드에서 T2 스레드를 생성 시 스레드는 3개가 되어 3개의 스레드가 일정 시간동안 문맥 교환을 하며 하나씩 실행된다.

이렇게 각 스레드는 일정 시간씩(2ms) 각각 실행되다 실행 시간이 끝나면 각각의 스레드는 소멸한다. 메인스레드가 먼저 소멸해서는 안 되며 다른 스레드가 모두 소멸될 때 메인 스레드가 종료 되며 메인 스레드가 종료되면 프로그램은 비로소 종료 된다

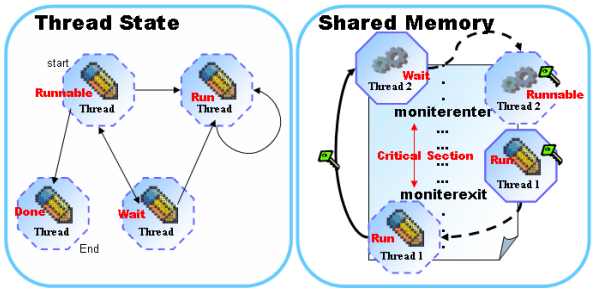


(그림 5) 스레드의 생성/소멸 흐름도

멀티 스레딩 과정에서 각각의 스레드는 공유하는 자원을 각각 접근하는 데에 문제가 발생하게 된다.

각각의 스레드는 상태를 가지며 실행대기(Runnable)상태, 실행상태(Run), 대기상태(Wait), 소멸상태(Done)로 이루어지며 흐름은 (그림 6)과 같다.

이 스레드의 상태와 키를 이용하여 공유 메모리의 접근 문제를 처리하며 (그림 6)과 같이 중간코드 상의 moniterenter와 moniterexit로 이루어져 있는 임계영역(Critical Section)코드를 처리 할 때 하나의 스레드가 하나의 키를 가져서 다른 스레드는 임계 영역에 접근하지 못하는 방식으로 처리한다. 스레드가 임계영역을 벗어나는 순간 키는 다른 스레드로 전달되며 그 스레드는 실행상태로 바뀌어 임계 영역을 실행하게 된다.[2]



(그림 6) 스레드의 상태와 공유메모리 처리 흐름도

4. 실험 결과 및 분석

u-VM의 멀티스레딩 기능을 검증하기 위해 Java 언어로 3개의 스레드를 생성하여 실험을 해 보았다. <표1>은 Java언어로 작성된 테스트 프로그램의 소스코드 이다.

<표 1> ThreadTest.java

```
class ThreadOne extends Thread{
        public void run(){
                int i = 0;
                while(i < 5){
                        System.out.println("Thread1");
                        i++;
                }
        }
}
class ThreadTwo extends Thread{
        public void run(){
                int i = 0;
                while(i < 5){
                        System.out.println("Thread2");
                        i++;
                }
        }
}
class ThreadThree extends Thread{
        public void run(){
                int i = 0;
                while(i < 5){
                        System.out.println("Thread3\n");
                        i++;
                }
        }
}
class ThreadTest{
        public static void main(String args[]){
                ThreadOne t1 = new ThreadOne();
                ThreadTwo t2 = new ThreadTwo();
                ThreadThree t3 = new ThreadThree();
                t1.run();
                t2.run();
                t3.run();
                System.out.println("MainThread");
        }
}
```

Java 소스코드를 Java Compiler를 통해 Bytecode로 변환 후 Bytecode-to-SAF 번역기를 통해 SAF 파일을 만들어 내

었다. <표2>은 번역된 SAF 파일의 일부이다.

<표 2> ThreadTest.saf 의 일부분

```
%%HeaderSectionStart
                    중간생략
   %EntryFunctionName &ThreadTest::main_([Ljava/lang/String;)V
%%HeaderSectionEnd
%%CodeSectionStart
   %FunctionStart
     .func_name &ThreadTest::main_([Ljava/lang/String;)V
                    중간생략
   %Line 40
            new         0           0
            dup
            str.p       1           16
            ldp
            lod.p       1           16
            call        &ThreadThree::<init>_()V
            str.p       1           12
   %Line 42
            ldp
            lod.p       1           4
            call        &ThreadOne::uVM_ThreadFunc()V
   %Line 43
            ldp
            lod.p       1           8
            call        &ThreadTwo::uVM_ThreadFunc()V
   %Line 44
                    중간생략
     .literal_start      @3   1    9
       0x54, 0x68, 0x72, . . . .
     .literal_end
   %LiteralTableEnd
   %InternalSymbolTableStart
   %InternalSymbolTableEnd
   %ExternalSymbolTableStart
   %ExternalSymbolTableEnd
%%DataSectionEnd
```

Java언어에서 스레드의 run()에 해당하는 이름은 B2S번역기에서 uVM_ThreadFunc()로 치환하여 줌으로 u-VM에서 식별가능

그리고 다음 <표3>는 SAF 파일을 u-VM에서 실행하기 위해 SAF 어셈블러를 통해 SEF로 나온 모습이다.

<표 3> ThreadTest.sef 의 일부분

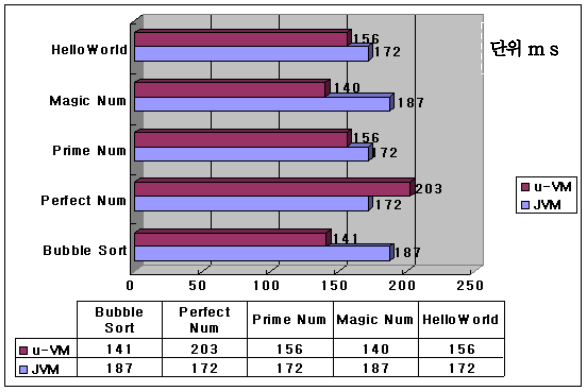
```
00000000h: 78 01 00 00 D5 01 00 00 D5 01 00 00 6C 02 00 00 ; x...?..?..?..
00000010h: FF FF FF FF 00 00 00 00 A4 00 04 00 00 00 01 00 ;            ....?..
00000020h: 01 00 28 00 F6 00 00 00 00 00 00 00 2E 00 01 00 ; ..(.?.........
00000030h: 00 00 00 00 A3 00 17 00 01 00 00 00 00 00 A2 00 ; ....?........?
00000040h: 24 01 00 00 00 00 00 00 2B 00 00 00 5D 00 00 00 ; $.......+...]..
00000050h: A3 00 A2 00 27 01 00 00 00 00 00 00 A3 00 A0 00 ; ??'.......??
00000060h: F6 00 00 00 00 00 00 00 A3 00 27 00 00 00 00 00 ; ?.......?'.....
00000070h: 00 00 09 00 04 00 00 00 A2 00 19 00 00 00 00 00 ; ........?......
00000080h: 00 00 A3 00 27 00 00 00 29 00 00 00 27 00 00 00 ; ..?'...)...'...
00000090h: 61 00 00 00 A2 00 53 00 00 00 00 00 00 00 09 00 ; a...?S.........
000000a0h: 00 00 00 00 5C 00 93 00 DA 00 00 00 A3 00 27 00 ; ....\.??..?'.
000000b0h: 00 00 2D 00 00 00 27 00 00 00 61 00 00 00 09 00 ; ..-...'...a....
000000c0h: 08 00 00 00 A2 00 21 00 00 00 00 00 00 00 A3 00 ; ....?!.......?
000000d0h: A2 00 25 01 00 00 00 00 00 00 A3 00 09 00 C8 00 ; ?%.......?..?
```

SEF를 u-VM에서 구동시킨 결과 (그림 6)과 같이 멀티스레딩이 잘 구동되는 것을 확인할 수 있다.

(그림 6) ThreadTest.sef 의 실행 화면

Java로 작성된 여러 개의 프로그램을 u-VM과 JVM에서 같이 구동 시켜 보고 ms단위로 실행 시간을 체크 하여 보았다.

예제프로그램은 Java의 여러 문법을 실험할 수 있는 프로그램을 택하였으며 (그림 7)와 같이 대부분의 프로그램이 JVM보다 약간 빠른 것을 볼 수 있다. Java의 클래스파일의 경우 최적화가 되어 있으나 현재 Bytecode-to-SAF 번역기는 최적화가 되어 있지 않다. 이를 가만해서라도 u-VM이 JVM보다 더 빠른 속도를 내는 것을 알 수 있다.



	Bubble Sort	Perfect Num	Prime Num	Magic Num	HelloWorld
u-VM	141	203	156	140	156
JVM	187	172	172	187	172

(그림 7) JVM과 u-VM의 실행속도 비교

5. 결론 및 향후 연구

본 논문에서는 유비쿼터스 게임 플랫폼의 멀티 스레드환경을 위한 가상기계인 u-VM을 설계하고 구현 하였다. u-VM의 구현으로 Java와 C, C++ 언어로 임베디드 콘텐츠를 제작하여 플랫폼에 구애받지 않고 임베디드 기기 상에서 콘텐츠를 실행할 수 있는 환경을 제공하게 되었다. 그리고 멀티스레딩 환경을 제공하는 스레드 스케줄러의 구현으로 멀티태스킹이 가능한 개발을 가능하게 하였다.

현재 u-VM은 그래픽 라이브러리를 추가하여 다양한 종류의 게임을 실행할 수 있도록 연구 중에 있으며 또한, u-VM을 PMP와 PDA에 포팅 중에 있다.

참고문헌

[1] Bill Venners, "Inside the JAVA Virtual Machine", McGraw-Hill,1999.

[2] Joshua Engel, "Programming for the Java™ Virtual Machine", Addison-Wesley, 1999

[3] Tim Lindholm, Frank Yellin, "The Java Virtual Machine Specification", Addison Wesley, 1999.

[4] Gagne, Silberschatz, Abraham "Operating System Concepts", Wiley, 2003.

[5] 이 양선, "임베디드 시스템을 위한 가상기계 기술", 한국멀티미디어학회지 Vol.6, No2, pp.36-44 Jun. 2002.

[6] Yang-Sun Lee, "Java Bytecode-to-.NET MSIL Translator for Construction of Platform Independent Information Systems," LNAI 3215, Vol.3, ISSN 0302-9743, Springer, pp.726-732, Sep 2004. .

[7] 최 홍석·김 영근·권 혁주·이 양선, "유비쿼터스 게임 플랫폼을 위한 임베디드 가상기계의 설계 및 구현", 한국정보처리학회, 2006 추계학술발표논문집, Vol.13, No.2, p.925, Nov 2006.